Verification of aggregated flows in OpenFlow networks

Sachin Sharma, Wouter Tavernier, Didier Colle, Mario Pickavet, and Piet Demeester
Department of Information Technology (INTEC), Ghent University - iMinds,
Email: {sachin.sharma, wouter.tavernier, didier.colle, mario.pickavet, piet.demeester}@intec.ugent.be

***Abstract*—Recently, the automatic test packet generation (ATPG) tool is proposed to verify a network for error conditions (e.g., incorrect firewall rules, software, hardware, and performance errors). However, this tool is not able to verify aggregated flows (i.e., flows having wildcards in some of their fields) for matching issues. In this paper, we propose a mechanism to verify aggregated flows in OpenFlow networks. In the demonstration, we verify aggregated flows installed in an emulated pan-European topology using our proposed mechanism.**

I. Introduction

In OpenFlow [1], the control plane is decoupled from switches or routers, and is embedded into one or more external servers called controllers. However, like traditional networks, debugging for errors (e.g., software, hardware, and firewall rules errors) is one of the time consuming and complex tasks in OpenFlow networks. One of the reasons of this complexity is the presence of aggregated flows, which contain several individual flows by having wildcards in some of their fields. As aggregated flows can match many different flows, it is difficult to find which flow is having an issue (such as matching issue). The issue could be caused by misconfiguration (e.g., firewall rules), software issues, hardware issues, or malicious attacks. Debugging all these errors is difficult by just analyzing the configuration of networks. Therefore, network operators have to debug manually by sending test packets in the network.

Manual debugging takes significant time in finding the issues. Therefore, the ATPG tool [2] is proposed recently for automatic verification of all flows by transmitting test packets in networks. However, this tool does not verify the matching header part of aggregated flows for software or hardware matching issues. In this paper, we propose a mechanism to verify all the aggregated flows for these issues. The aim of our mechanism is : (1) to perform automatic discovery of matching errors, (2) to minimize the time required to find errors, (3) to minimize the bandwidth required for verification. The mechanism is implemented in the part of a verification activity [3] in the UNIFY project (www.fp7-unify.eu).

For the demo, we first generate the matching errors for aggregated flows and then find these errors using our mechanism. Executing the mechanism, generating errors, and finding the errors are performed by different GUIs (Graphical User Interface) placed in the controller and in an emulated switch topology (pan-European topology). In addition, during the demo, we transmit data traffic containing faulty and non-faulty flows, and show that the errors found through our mechanism are actually present in the network, as traffic containing faulty flows are not received by the destinations.

II. Verification mechanism

In our mechanism, for verification, two steps – (1) flow-entry transformation and (2) test packet generation – are performed. The flow-entry transformation step transforms an aggregated flow-entry into three entries and can be performed at the time of flow addition. The test packet generation step verifies these three entries for matching issues and can be performed at any time when the verification of the aggregated flow is required.

In the mechanism, a field (such as EtherType or VLAN) of a flow is used to distinguish data and test packets. It is assumed that this field is always wildcarded for data flows. This assumption allows the mechanism to add the same matching-header for two of the entries (flow-entry 1 and 2 in Fig. 1) in the flow-entry transformation step. As the matching-header part of these entries is same, the assumption is that if a matching error is present in one entry, it is also present in the other entry.

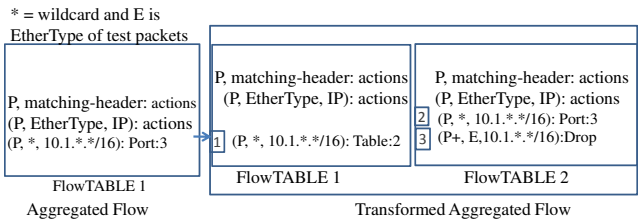

Fig. 1. Flow-Entry Transformation step. P+ is a priority no. higher than P

The flow-entry transformation step is shown in Fig .1. In this step, an aggregated flow is transformed into three flow-entries. The first entry is present in the same FlowTable as the original flow, and the other two entries are present in another FlowTable i.e., FlowTable 2 in Fig .1. The first flow-entry, which has the same matching header as the original flow, redirects all the matched packets to another table (i.e., FlowTable 2). The second flow-entry, which has same matching-header as the first entry and a lower priority number than the third entry, redirects all the matched data packets to the output action (Port:3) of the original flow. The third flow-entry, which contains EtherType (E) of test packets in its matching header part, drops all the matched test packets. After establishing these entries, the controller can now perform verification using test packet generation step.

In the test packet generation step, the controller finds the matching errors in the aggregated flow by transmitting test packets. For finding errors in a switch, the controller in our mechanism sends first all the test packets (which can be matched through an aggregated flow). It then finds the number of errors by subtracting the number of sent packets from the increase in the counters (statistics) of the third flow entry (the third entry) in FlowTable 2. It also finds that if there is an increase in counters of another test packet flow-entry containing unmatched flow (due to incorrect matching). At this time, the controller knows the number of flow matching errors. However, it does not know that which flows have a

matching issue. The controller finds these errors by applying the binary search algorithm. In this algorithm, the controller now transmits the first half of the test packets that the Flow Entry can match and then finds the number of errors in these sent packets by the same formula (i.e., subtracting the number of sent packets from the counter increments) and checking counters increment in another Flow Entries. If no error is present, it transmits the other half of the test packets and finds errors. This process is repeated until all the flow matching errors (incorrect or no matching) are found.

Our mechanism provides the correct matching errors if the increase in the counters of the first entry is equal to the sum of the increase in the counters of the second and third entries.

III. Experiments and Results

In this section, we describe a verification experiment performed on the OFELIA testbed [4] using an emulated topology shown in Fig. 3. The topology contains 16 switches and makes an out-of-band connection with the controller. Each switch is installed with Open vSwitch software [5] and OpenDayLight [6] together with our mechanism is installed in the controller. The counter update interval of flow-entries is 3 seconds in Open vSwitch. In addition, the bandwidth shown in X-axis of Fig. 2 is dedicated to a link between the controller and switches for transmitting/receiving test packets.

We add 10 aggregated flows in each switch and each aggregated flow can match 2^{16} different flows. For generating matching errors in aggregated flows, we manually added firewall rules so that 2^8 different flows (out of 2^{16} flows) can not match through an aggregated flow. In our experiment, we find these unmatched flows through our verification mechanism using 9 iterations of the binary search algorithm (Section II).

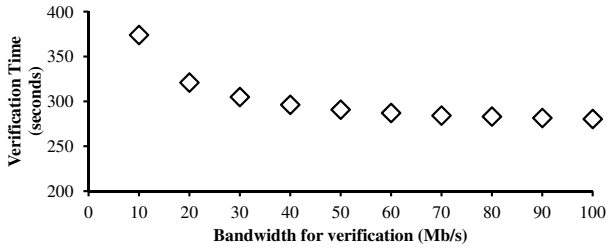

Fig. 2. Time Spent in Verification

Fig. 2 shows that the time spent in verification with the bandwidth used for verification.

IV. Demonstration on Portable Testbed

With the portable testbed (2 laptops, Fig. 3), we show the working of our proposed verification mechanism using an emulated pan-European topology. The emulated topology is generated in the lower laptop (shown in Fig. 3) using Mininet and the controller (OpenDayLight) with our mechanism is configured in the upper laptop. The connection between two laptops is done through an Ethernet cable.

For the demonstration, aggregated flows are pre-installed in the emulated topology. In addition, error conditions are generated by applying firewall rules for some flows of aggregated flows.

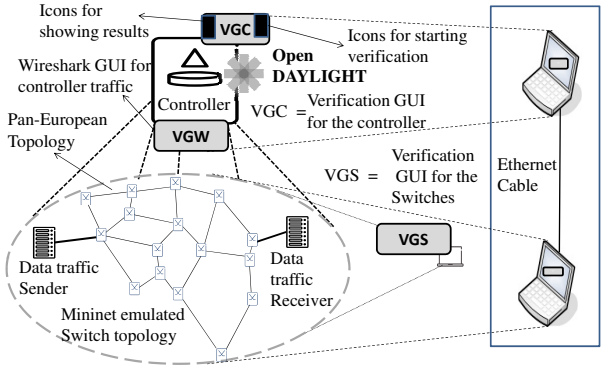

Fig. 3. Live Demonstration Scenario

In the demonstration, we show the working of the verification mechanism using three verification GUIs (VGC, VGW, and VGS), shown in Fig. 3. The controller GUI, VGC, is the extension of the GUI implemented for the OpenDayLight controller [6] and is shown in the upper laptop. In this GUI, new icons are implemented for starting verification and showing the results (See Fig. 3). The wireshark GUI, VGW, is the GUI of wireshark (well known tracing tool) to show the test packets traveling from the controller as packet out messages. The switch GUI, VGS, is the GUI to show the bandwidth usage by test packets using our verification mechanism. The dark bold color shows the bandwidth usage by test packets and the light color shows that no test packet is flowing at that time for verification.

In the demonstration, we start verification of aggregated flows by clicking the implemented icons in the VGC GUI (See Fig. 3) and the controller then starts verification by transmitting test packets, as explained in Section II. These packets travel from the controller to the emulated topology through the Ethernet cable.

When the verification mechanism is completed, the results are shown in the icons of GUI (shown in Fig. 3). The results show the unmatched flows (i.e., flows that cannot be matched through an aggregated flow), the number of transmitted test packets for finding errors, and the time taken by our mechanism to find the results.

In addition to these GUI demonstrations, we manually transmit the data packets from one of the switches (as shown in Fig. 3) and show that data packets representing the unmatched flows shown in the VGC GUI are actually not received by the destination and hence, the errors are present in the network.

Acknowledgment

This research has received funding from the EU FP7 under agreement n^o 619609 (Unify).

References

[1] N. McKeown et al., Openflow: Enabling innovation in campus networks, ACM Computer Communication Review, Vol. 38, pp. 69–74 2008

[2] H. Zeng et. al., Automatic Test Packet Generation, IEEE/ACM Transactions on Networking, Vol. 22, pp. 554-566, 2013

[3] J. Kim et. al., Service Provider DevOps for Large Scale Modern Network Services, BDIM, 2015

[4] OFEILA testbed: http://www.fp7-ofelia.eu/

[5] Open vSwitch: http://openvswitch.org/

[6] OpenDayLight: http://www.opendaylight.org/